Deciding Definability by Deterministic Regular Expressions

Wojciech Czerwiński, Claire David, Katja Losemann, Wim Martens

Deciding Definability by Deterministic Regular Expressions*

Wojciech Czerwiński[1], Claire David[2], Katja Losemann[1], and Wim Martens[1]

[1] Universität Bayreuth
[2] Université Paris-Est Marne-la-Vallée

Abstract. We investigate the complexity of deciding whether a given regular language can be defined with a deterministic regular expression. Our main technical result shows that the problem is PSPACE-complete if the input language is represented as a regular expression or nondeterministic finite automaton. The problem becomes EXPSPACE-complete if the language is represented as a regular expression with counters.

1 Introduction

Schema information is highly advantageous when managing and exchanging XML data. Primarily, schema information is crucial for automatic error detection in the data itself (which is called validation, see, e.g., [5, 26, 2, 20]) or in the procedures that transform the data [24, 23, 22]. Furthermore, schemas provide information for optimization of XML querying and processing [25, 28], they are inevitable when integrating data through schema matching [1], and they provide users with a high-level overview of the structure of the data. From a software development point of view, schemas are very useful to precisely specify pre- and post-conditions of software routines that process XML data.

In their core, XML schemas specify the structure of well-formed XML documents through a set of constraints which are very similar to extended context-free grammar productions. Such schema are usually abstracted as a set of rules of the form

$$\mathit{Type} \to \mathit{Content},$$

where *Content* is a regular expression that defines the allowed content inside the element type specified in the left-hand side. As such, regular expressions are a central component of schema languages for XML.

The two most prevalent schema languages for XML data, *Document Type Definition (DTD)* [5] and *XML Schema Definition (XSD)* [10], both developed by the World Wide Web Consortium, do not allow arbitrary regular expressions to define *Content*. Instead, they require these expressions to be *deterministic*. We refer to such deterministic regular expressions as *DREs*. In order to get a good understanding of schema languages for XML, it is thus important to develop a

* This work was supported by grant number MA 4938/2-1 of the Deutsche Forschungsgemeinschaft (Emmy Noether Nachwuchsgruppe).

good understanding on DREs. Furthermore, since the concept of determinism in regular expressions is a rather foundational, we believe our results to be relevant in a larger scope as well.

Intuitively, a regular expression is deterministic if, when reading a word from left to right without looking ahead, it is always clear where in the expression the next symbol can be matched. For example, the expression $(a + b)^*b(a + b)$ is not deterministic, because if we read a word that starts with b, it is not clear whether this b should be matched in the expression if we do not know what the remainder of the word will be. As such, determinism in regular expressions is very similar to determinism in finite automata: When we consider each alphabet symbol in an expression as a state and consider transitions between positions in the expression that can be matched by successive symbols, then the expression is deterministic if and only if the thus obtained automaton (which is known as the Glushkov automaton of the expression) is deterministic.

Deterministic regular expressions or DREs have therefore been a subject of research since their foundations were laid in a seminal paper by Brüggemann-Klein and Wood [6, 7]. The most important contribution of this paper is a characterization of languages definable by DREs in terms of structural properties on the minimal DFA. In particular, this characterization showed that some regular languages cannot be defined with a DRE. One such language is defined by the expression $(a + b)^*b(a + b)$. Furthermore, Brüggemann-Klein and Wood showed that it is decidable whether a given regular language is definable by a DRE. Since then, DREs have been studied in the context of language approximations [3], learning [4], descriptional complexity [13, 21] and static analysis [8, 9]. Recently, it was shown that testing if a regular expression is deterministic can be done in linear time [14].

Determinism has also been studied for a more general class of regular expressions which allows a *counting operator* [19, 11, 16]. This operator allows to write the expression $a^{10,100}$ defining the language that contains strings of length 10 to 100 and labeled with only a's. The motivation for the counting operator again comes from schema languages, because the operator can be used to define expressions in XML Schema. Determinism for expressions with counters seems to pose more challenges than without the counting operator. For example, already testing whether an expression with counters is deterministic is non-trivial [18].

In this paper we study the following problem:

Given a regular expression, can it be determinized?

This problem seems to be very foundational and has first been studied around 20 years ago [6] but the precise complexity was still open, despite the rich body of research discussed above. The best known upper bound is from Brüggeman-Klein and Wood, who showed that the problem is in EXPTIME (by exhibiting an algorithm that works in polynomial time on the minimal DFA [7]) and the best known lower bound is PSPACE-hardness [3]. The main result of this paper settles this question and proves that this problem is PSPACE-complete. Our proof is rather technical and provides deeper insights in the decision algorithm of Brüggemann-Klein and Wood. A central insight, which is a cornerstone of our

proof, is that the recursion depth of the algorithm is only polynomial in the size of the smallest NFA for the given regular language.

Since regular expressions with counters are important in the context of W3C XML Schema, we also study the complexity of deciding if a given expressions with counters can be written as a DRE. This problem turns out to be EXPSPACE-complete. We complement these completeness results by proving that it is NLOGSPACE-hard to decide if a given DFA can be written as a DRE. At the moment, it is not clear to us whether this lower bound can be improved. The problem is known to be in polynomial time by [7].

Organisation: We give the basic definitions in Section 2. In Section 3 we present the algorithm of Brüggeman-Klein and Wood and prove preliminary results. Complexity results are presented in Section 4. Due to space limits, some proofs are not presented or only sketched.

2 Definitions

For a finite set S, we denote its cardinality by $|S|$. By Σ we always denote an alphabet, i.e., a finite set of symbols. A (Σ-)*word* w over alphabet Σ is a finite sequence of symbols $a_1 \cdots a_n$, where $a_i \in \Sigma$ for each $i = 1, \ldots, n$. The set of all Σ-words is denoted by Σ^*. The *length* of a word $w = a_1 \cdots a_n$ is n and is denoted by $|w|$. The empty word is denoted by ε. A *language* is a set of words.

A *(nondeterministic) finite automaton* (or *NFA*) N is a tuple $(Q, \Sigma, \delta, q_0, F)$, where Q is a finite set of states, $\delta : Q \times \Sigma \to 2^Q$ is the transition function, q_0 is the initial state, and $F \subseteq Q$ is the set of accepting states. We sometimes denote that $q_2 \in \delta(q_1, a)$ as $q_1 \xrightarrow{a} q_2 \in \delta$ to emphasize that, when N is in state q_1, it can go to state q_2 when reading an a. A *run of N on word $w = a_1 \cdots a_n$* is a sequence $q_0 \cdots q_n$ where, for each $i = 1, \ldots, n$, we have $q_{i-1} \xrightarrow{a_i} q_i \in \delta$. Word w is *accepted* by N if there is such a run which is *accepting*, i.e., if $q_n \in F$. The *language of N*, also denoted $L(N)$, is the set of words accepted by N. By δ^* we denote the extension of δ to words, i.e., $\delta^*(q, w)$ is the set of states which can be reached from q by reading w. The *size* $|N|$ of an NFA is the total number of transitions, i.e., $\sum_{q,a} |\delta(q, a)|$. An NFA is *deterministic*, or a *DFA*, when every $\delta(q, a)$ has at most one element. Throughout the paper, we will use the notation $\mathcal{P}_N$ for the power set automaton of N and $[N]$ for the minimal DFA for $L(N)$. It is well-known that $[N]$ is unique for N and that it can be obtained by merging states of $\mathcal{P}_N$ [15]. In this paper we assume that all states of automata are *useful* unless mentioned otherwise, that is, every state can appear in some accepting run. This implies that every state can be reached from the initial state and that, from each state in an automaton, an accepting state can be reached. This also implies that we use minimal DFAs without sink state and that $\mathcal{P}_N$ by default only contains the useful subsets of states of N. We sometimes abuse notation and also denote by $\emptyset$ the minimal DFA with no states.

Furthermore, we will often see an NFA as a *graph*, which is obtained by considering its states as nodes and its transitions as (labeled) directed edges. Then, we also refer to a connected sequence of transitions in N as a *path*.

The *regular expressions (RE)* over Σ are defined as follows: ε and every Σ-symbol is a regular expression; and whenever r and s are regular expressions then so are $(r \cdot s)$, $(r + s)$, and $(s)^*$. In addition, we allow $\emptyset$ as a regular expression, but we do not allow $\emptyset$ to occur in any other regular expression. For readability, we usually omit concatenation operators and parentheses in examples. The *language* defined by an RE r, denoted by $L(r)$, is defined as usual. Whenever we say that expressions or automata are *equivalent*, we mean that they define the same language. The *size* $|r|$ of r is defined to be the total number of occurrences of alphabet symbols, epsilons, and operators, i.e., the number of nodes in its parse tree.

2.1 Variations of Regular Expressions

The *regular expressions with counters (RE(#))* extend the REs with a *counting operator*. That is, each RE-expression is an RE(#)-expression. Furthermore, when r and s are RE(#)-expressions then so are $(r \cdot s)$, $(r + s)$, and $r^{k,\ell}$ for $k \in \mathbb{N}$ and $\ell \in \mathbb{N}^+ \cup \{\infty\}$ with $k \leq \ell$. Here, $\mathbb{N}^+$ denotes $\mathbb{N} \setminus \{0\}$. For a language L, define $L^{k,\ell}$ as $\bigcup_{i=k}^{\ell} L^i$. Then, $L(r^{k,\ell}) = \bigcup_{i=k}^{\ell} L(r)^i$. Notice that r^* is equivalent to $r^{0,\infty}$. The size of an expression in RE(#) is the number of nodes in its parse tree, plus the sizes of the counters, where a counter $k \in \mathbb{N}$ has size $\log k$.

Deterministic regular expressions (DREs) put a restriction on the class of REs. Let $\bar{r}$ stand for the RE obtained from r by replacing, for every i and a, the i-th occurrence of alphabet symbol a in r (counting from left to right) by a_i. For example, for $r = b^*a(b^*a)^*$ we have $\bar{r} = b_1^*a_1(b_2^*a_2)^*$. A regular expression r is *deterministic* (or *one-unambiguous* [7] or a *DRE*) if there are no words wa_iv and wa_jv' in $L(\bar{r})$ such that $i \neq j$. The expression $(a + b)^*a$ is not deterministic since both strings a_2 and a_1a_2 are in $L((a_1 + b_1)^*a_2)$. The equivalent expression $b^*a(b^*a)^*$ is deterministic. Brüggemann-Klein and Wood showed that not every regular expression is equivalent to a deterministic one [7]. We call a regular language *DRE-definable* if there exists a DRE that defines it. The canonical example for a language that is not DRE-definable is $(a + b)^*b(a + b)$ [7]. We therefore have that the set of DREs forms a strict subset of the REs, which in turn are a strict subset of the RE(#)s.

2.2 Problems of Interest

In this paper, we investigate variants of the following problem.

DRE-Definability: Given a regular language L, is L DRE-definable?

We consider this problem for various representations of regular languages: regular expressions, regular expressions with counters, NFAs, and DFAs. Whenever we consider such a variation, we put the respective representation between braces. For example, DRE-Definability(RE) is the problem: Given a regular expression r, is $L(r)$ DRE-definable?

3 The BKW Algorithm

DRE-Definability was first studied by Brüggemann-Klein and Wood who showed that the problem can be solved in polynomial time in the size of the minimal DFA of a language [7]. Their algorithm (henceforth referred to as the Bkw-Algorithm) is not at all trivial and gives good insight in DRE-definable regular languages. We recall the Bkw-Algorithm together with some definitions and known results and then we prove deeper properties of the Bkw-Algorithm which will be the basis of further results in the paper.

Orbits and gates: For a state q in an NFA N, the *orbit of* q, denoted $\mathcal{O}(q)$, is the maximal strongly connected component of N that contains q. We call q a *gate* of $\mathcal{O}(q)$ if q is accepting or q has an outgoing transition that leaves $\mathcal{O}(q)$. If an orbit consists only of one state q and q has no self-loops, we say that it is a *trivial* orbit. We say that a transition $q_1 \xrightarrow{a} q_2$ is an *inter-orbit* transition if q_1 and q_2 belong to different orbits. The *orbit automaton of state* q is the sub-automaton of N consisting of $\mathcal{O}(q)$ in which the initial state is q and the accepting states are the gates of $\mathcal{O}(q)$. We denote the orbit automaton of q by N_q. The *orbit language of* q is $L(N_q)$. The *orbit languages of* N are the orbit languages of states of N.

Orbit property: An NFA N has the *orbit property* if, for every pair of gates q_1, q_2 in the same orbit in N, the following properties hold:

1. q_1 is accepting if and only if q_2 is accepting; and,
2. for all states q outside the orbit of q_1 and q_2, there is a transition $q_1 \xrightarrow{a} q$ if and only if there is a transition $q_2 \xrightarrow{a} q$.

Consistent symbols: A symbol $a \in \Sigma$ is *N-consistent* if there is a state $f(a)$, such that every accepting state q of N has a transition $q \xrightarrow{a} f(a)$. We refer to the corresponding transitions as *consistent transitions* of N. A set $S \subseteq \Sigma$ is *N-consistent* if every symbol in S is N-consistent. Whenever we consider N-consistent sets S in the remainder of the paper we assume that they are maximal, i.e., there does not exist an $a \in \Sigma$ that is not in S and is N-consistent. Henceforth, we will therefore refer to *the* N-consistent set. For the set S of N-consistent symbols, the *S-cut* of N, denoted N_S, is obtained by removing all consistent transitions from N. Using these notions, Brüggemann-Klein and Wood give the following characterization of the class of DRE-definable languages.

Theorem 1 (Brüggemann-Klein and Wood [7]). *Let D be a minimal DFA and S be the set of D-consistent symbols. Then the following are equivalent:*

1. *$L(D)$ is DRE-definable;*
2. *D has the orbit property and all orbit languages of D are DRE-definable;*
3. *D_S has the orbit property and all orbit languages of D_S are DRE-definable.*

Furthermore, if D consists of a single, nontrivial orbit and $L(D)$ is DRE-definable, then there is at least one D-consistent symbol.

Algorithm 1 The Bkw-Algorithm [7].

```
   Algorithm Bkw
 2: Input: Minimal DFA D = (Q, Σ, δ, q0, F)
   Output: true if L(D) is DRE-definable, else false
 4: S ← the maximal set of D-consistent symbols
   if D has only one trivial orbit then return true
 6: if D has precisely one orbit and S = ∅ then return false
   compute the orbits of D_S
 8: if D_S does not have the orbit property then return false
   for each orbit O in D_S do
10:     choose a state q in O
        if not Bkw((D_S)_q) then return false
12: return true
```

They also show this result about the orbit property and the orbit languages.

Lemma 2 (Brüggemann-Klein and Wood [7]). *Let D be a minimal DFA and S be the set of D-consistent symbols.*

1. *If D_S has the orbit property, then $(D_S)_q$ is minimal for each state q in D.*
2. *If p and q are states in the same orbit of D_S, then $L((D_S)_p)$ is DRE-definable if and only if $L((D_S)_q)$ is DRE-definable.*

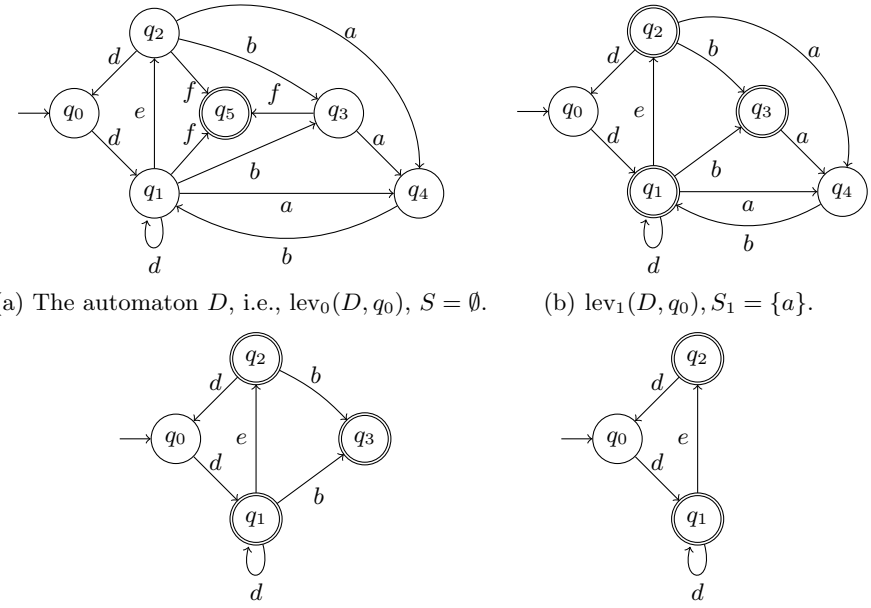
Point 1 of the above lemma is immediate from combining Lemmas 5.9 and 5.10 from [7]. Point 2 is immediate from the fact that DRE-definable regular languages are closed under derivatives [7]. Notice that, in general, D_S does not have to be a minimal DFA. In particular, it can have states that are not reachable from the initial state. These results lead to a recursive test that decides whether the language of a minimal DFA is DRE-definable. We present this test in Algorithm 1. Notice that Lemma 2 ensures that we never have to minimize the DFA that we give to the recursive call in line 11 of the algorithm.

In the remainder of this article, D always denotes a minimal DFA. In the following we investigate the recursion depth of Algorithm 1. Therefore, we examine how, for a state q of D, the orbit of q evolves during the recursion. In one iteration of Algorithm 1 we always delete two kinds of transitions, if they are present: the consistent transitions (which we delete to obtain D_S from D) and the inter-orbit transitions in D_S (which we delete to obtain $(D_S)_q$).

Level automata: For a state q of a minimal DFA D and $k \in \mathbb{N}$ we inductively define the *level k automaton of D for the state q*, denoted $\mathrm{lev}_k(D, q)$, as follows:

- $\mathrm{lev}_0(D, q) = D$.
- Let S be the maximal set of consistent symbols in D. Then

$$\mathrm{lev}_1(D,q) = \begin{cases} (D_S)_q & \text{if } D \text{ has more than one orbit and } D_S \text{ has the orbit property;} \\ (D_S)_q & \text{if } S \neq \emptyset \text{ and } D_S \text{ has the orbit property;} \\ \emptyset & \text{otherwise.} \end{cases}$$



(a) The automaton D, i.e., $\mathrm{lev}_0(D, q_0)$, $S = \emptyset$.

(b) $\mathrm{lev}_1(D, q_0), S_1 = \{a\}$.

(c) The S_1-cut of $\mathrm{lev}_1(D, q_0)$.

(d) $\mathrm{lev}_2(D, q_0), S_2 = \emptyset$.

Fig. 1. An example of level automata for a minimal DFA D.

– Let $k > 1$ and S_{k-1} be the maximal set of consistent symbols in $B := \mathrm{lev}_{k-1}(D, q)$. Then

$$\mathrm{lev}_k(D, q) = \begin{cases} (B_{S_{k-1}})_q & \text{if } S_{k-1} \neq \emptyset \text{ and } B_{S_{k-1}} \text{ fulfills the orbit property} \\ \emptyset & \text{otherwise.} \end{cases}$$

The above definition actually precisely follows the construction in Algorithm 1 if state q is chosen every time in line 10. The definition makes clear that the top level recursion of the Bkw-Algorithm (in which we construct $\mathrm{lev}_1(D, q)$) is slightly different from the others: the input DFA D of the top level can have multiple orbits, whereas this is not the case for deeper recursive levels. According to Lemma 2, $\mathrm{lev}_k(D, q)$ is always minimal.

Example 3. Figure 1 provides an example to illustrate the notion of level automata. Consider the minimal DFA D from Figure 1(a) and its state q_0. By definition, $\mathrm{lev}_0(D, q_0)$ is the automaton D itself. In order to build $\mathrm{lev}_1(D, q_0)$ (see Figure 1(b)), observe that D has two orbits and its set of consistent symbols S is empty since no transitions leave state q_5. Furthermore, D_S, which equals D, fulfills the orbit property since all transitions that leave $\mathcal{O}(q_0)$ go to state q_5. As such, $\mathrm{lev}_1(D, q_0)$ equals $(D_\emptyset)_{q_0}$, the orbit automaton of q_0 in D. We now

explain how to obtain $\mathrm{lev}_2(D, q_0)$ (see Figure 1(d)). First notice that $S_1 = \{a\}$ is the maximal set of consistent symbols in $\mathrm{lev}_1(D, q_0)$. Furthermore, the S_1-cut of $\mathrm{lev}_1(D, q_0)$ (illustrated in Figure 1(c) without unreachable states) fulfills the orbit property. The automaton $\mathrm{lev}_2(D, q_0)$ is the orbit automaton of q_0 in the S_1-cut of $\mathrm{lev}_1(D, q_0)$ (that is, $\mathrm{lev}_2(D, q_0) = (\mathrm{lev}_1(D, q_0)_{S_1})_{q_0}$). Finally, observe that $\mathrm{lev}_2(D, q_0)$ has only one orbit and no consistent symbols which implies that $\mathrm{lev}_3(D, q_0) = \emptyset$. Also, in accordance with the Bkw-Algorithm, this means that $L(D)$ is not DRE-definable.

The following lemma summarizes the link between DRE-definability and level automata.

Lemma 4. *Let D be a minimal DFA. Then the following are equivalent:*

1. *$L(D)$ is DRE-definable;*
2. *for every state q of D and $k \in \mathbb{N}$, $L(lev_k(D, q))$ is DRE-definable;*
3. *for every state q of D and $k \in \mathbb{N}$, $L(lev_k(D, q))$ is DRE-definable and $lev_k(D, q)_{S_k}$ has the orbit-property.*

3.1 The Recursion Depth of BKW

First we observe that, once a state becomes a gate in the Bkw-Algorithm, its outgoing transitions will disappear in deeper recursion levels.

Lemma 5. *Let D be a minimal DFA and q be a gate in $lev_k(D, q)$ for some $k > 0$. Then either $lev_{k+1}(D, q) = \emptyset$ or q has strictly less outgoing transitions in $lev_{k+1}(D, q)$. In the latter case, q is also a gate in $lev_{k+1}(D, q)$.*

From Lemma 5, we can infer how long it takes for a state p to become a gate.

Lemma 6. *Let D be a minimal DFA and p be a state of $lev_k(D, p)$ for some $k \in \mathbb{N}$. Let ℓ be the length of the shortest path from p to a gate in $lev_k(D, p)$. Then either $lev_{k+|\Sigma|\cdot\ell+1}(D, p) = \emptyset$ or p is a gate in $lev_{k+|\Sigma|\cdot\ell+1}(D, p)$.*

Next, we want to combine Lemma 6 with an observation about NFAs versus their minimal DFAs, namely that paths to accepting states are short.

Lemma 7. *Let N be an NFA with size n. Then for every state of $[N]$ there is a path leading to some accepting state of length at most $n - 1$.*

Combining Lemma 6 and 7 tells us how long states can be present in the recursion of the Bkw-Algorithm, compared to the size of an NFA for the language.

Lemma 8. *Let N be an NFA with size n. Then it holds that $lev_{n\cdot|\Sigma|+2}([N], p) = \emptyset$ for every state p of $[N]$.*

Summarized, we know that the recursion depth of the Bkw-Algorithm is polynomial in the size of the minimal NFA for a language.

Theorem 9. *Let N be an NFA with size n. The recursion depth of Algorithm 1 on $[N]$ is at most $n \cdot |\Sigma| + 2$.*

3.2 Consistency Violations

In the following, we analyze the possible causes of failure for the Bkw-Algorithm. We identify three properties such that the Bkw-Algorithm fails if and only if one of them holds for some orbit automaton at some level k. This will be a tool for our PSPACE algorithm that will search for one of these violations.

From the Bkw-Algorithm, we can immediately see that there are two situations in which it can reject: (in line 6) at some point in the recursion, the automaton consists only of one orbit which has no consistent symbols or (in line 8) at some point in the recursion, the S-cut of the automaton does not have the orbit property. The latter means that there exist two gates of the same orbit in the S-cut, such that either they do not have the same transitions to the outside or one of them is accepting while the other one is not. We now formalize these different types of violations and we then prove that the Bkw-Algorithm fails if and only if one of these violations is found at some point in the recursion. Let D be a minimal DFA and S be its set of consistent symbols. Then D has

- an Out-Consistency Violation, if there exist gates q_1 and q_2 in the same orbit $\mathcal{O}$ of D_S and there exists a state q outside $\mathcal{O}$ such that there is a transition $q_1 \xrightarrow{a} q$ and no transition $q_2 \xrightarrow{a} q$;
- an Acceptance Consistency Violation, if there exist gates q_1 and q_2 in the same orbit of D_S such that q_1 is accepting and q_2 is not; and
- an Orbit Consistency Violation, if there exists an accepting state q_1 such that, for every symbol a, there exists another accepting state q_2 in $\mathcal{O}(q_1)$ in D, such that for every state q, at most one of the transitions $q_1 \xrightarrow{a} q$ and $q_2 \xrightarrow{a} q$ exists.[3]

Notice that the first two violations focus on D_S and the last one on D, as in the Bkw-Algorithm. In summary, we will also say that a DFA D *has a violation* if and only if it has at least one of the above violations. We show that these violations are a valid characterization of DRE-definable languages.

Theorem 10. *Let D be a minimal DFA. Then it holds that $L(D)$ is not DRE-definable if and only if there exist a state q of D and $k \in \mathbb{N}$ such that $lev_k(D, q)$ has a violation.*

4 The Definability Problem

We are now ready to prove results about the complexity of DRE-Definability for different formalisms. We first show that the problem is PSPACE-complete for NFAs and REs. Then we look at RE(#)s which are natural extensions in the context of W3C XML Schema. In that setting, the problem becomes EXPSPACE-complete. Finally, we look at the class of DFAs.

[3] Notice that $\delta(q_1, a)$ may be empty if D only has useful states.

4.1 Definability for REs and NFAs

Our PSPACE algorithm for DRE-definability for REs and NFAs exploits Theorem 10 in the following way. Given an NFA N, we search for a level k and a state p of $[N]$ such that $\mathrm{lev}_k([N], p)$ has a violation. As PSPACE is closed under complement, the result follows.
Notice that, in general $[N]$ can be exponentially larger than N and therefore we cannot simply compute $[N]$ in space polynomial in $|N|$. To overcome this difficulty, we use the following two ideas:

1. Use the fact that the maximal recursion depth of Algorithm 1 on $[N]$ is polynomial in the size of the NFA N (Theorem 9);
2. Adapt Algorithm 1 using Theorem 10 and apply it on the minimal DFA by only partially constructing it on-the-fly from the NFA.

In the following we explain how we can detect if there occurs a violation in the minimal DFA for some NFA on the fly, i.e., without constructing the DFA explicitly. To this end, we fix the following notations for the remainder of the section. By $N = (Q_N, \Sigma, \delta_N, q_N^0, F_N)$ we always denote an NFA. So, in particular, we always denote by Q_N the state set of N. For a set of states $q \subseteq Q_N$, we denote by $[q]$ the corresponding state in the minimal DFA $[N]$. More formally, $[q]$ is the set of words $\{w \mid \exists t \in q \text{ s.t. } \delta_N^*(t, w) \cap F_N \neq \emptyset\}$, i.e., the Myhill-Nerode class of q. Also, whenever we talk about $\mathrm{lev}_k([N], [q])_{S_k}$, the set S_k is the set of consistent symbols in $\mathrm{lev}_k([N], [q])$. The key result (Lemma 17) is to show that we can detect if a violation occurs in a level k for $[N]$ in space polynomial in k and $|N|$. Here are the precise problems we consider. For each of them the input is an NFA N and $k \in \mathbb{N}$:

Out-Cons-Violation:	Given N and k, is there a $q \subseteq Q_N$ such that $\mathrm{lev}_k([N], [q])_{S_k}$ has an out-consistency violation?
Acc-Cons-Violation:	Given N and k, is there a $q \subseteq Q_N$ such that $\mathrm{lev}_k([N], [q])_{S_k}$ has an acceptance consistency violation?
Orbit-Cons-Violation:	Given N and k, is there a $q \subseteq Q_N$ such that $\mathrm{lev}_k([N], [q])$ has an orbit consistency violation?

We first study the complexity of the following subproblems which will be used in the proof of Lemma 17. The input is always a subset of an NFA N, sets $p, q \subseteq Q_N$, $a \in \Sigma$, $k \in \mathbb{N}$ that is relevant to the problem.

Edge:	Given (N, p, q, a, k), is $[p] \xrightarrow{a} [q]$ a transition in $\mathrm{lev}_k([N], [p])$?
Reachability:	Given (N, p, q, k), is $[q]$ reachable from $[p]$ in $\mathrm{lev}_k([N], [p])$?
SameOrbit:	Given (N, p, q, k), are $[p]$ and $[q]$ in the same orbit of $\mathrm{lev}_k([N], [p])$?
InterOrbit:	Given (N, p, k), is there an inter-orbit transition $[p] \xrightarrow{a} [q]$ for some label a and q in $\mathrm{lev}_k([N], [p])$?
Acceptance:	Given (N, p, k), is $[p]$ accepting in $\mathrm{lev}_k([N], [p])$?
IsGate:	Given (N, p, k), is $[p]$ a gate in $\mathrm{lev}_k([N], [p])$?

Notice that SameOrbit and InterOrbit are only non-trivial if $k = 0$. Furthermore, for some of the above problems X we consider a variation called X-Cut in which, with the same input, we want to decide if the problem X is true for automaton $\mathrm{lev}_k([N], [p])_{S_k}$ (instead of $\mathrm{lev}_k([N], [p])$).
We will heavily use the following result:

Theorem 11 (Corollary to Savitch's Theorem). *Let $f(n) \geq \log n$ be a non-decreasing polynomial function. Then NSPACE($f(n)$) $\subseteq$ SPACE($f^2(n)$).*

Our proof is a careful mutual induction on the above defined problems. First we show that Edge, Edge-Cut, and Acceptance can be computed in polynomial space on level 0 and then we prove a set of implications of the sort *if we can solve X on level k, then we can solve Y on level k (or level $k+1$).* All the lemmas have to be carefully put together in the right order.

Lemma 12. *Given N and $p, q \subseteq Q_N$, we can test whether $[p] = [q]$ in space $O(|N|^2)$.*

Proof (Sketch). A non-deterministic Turing Machine can test in non-deterministic space $O(|N|)$ whether $[p] \neq [q]$. The lemma then follows from Theorem 11 and the fact that deterministic complexity classes are closed under complement. □

In the following, whenever we say that we solve a problem *for N at level k*, we mean that we solve it for the NFA N and arbitrary sets of states $p, q \subseteq Q_N$, $a \in \Sigma$, and k. The basis of our entire mutual induction is being able to test if a certain transition is present in the minimal DFA equivalent to N, if it is present in its S-cut, or if some state is accepting.

Lemma 13. Edge, Acceptance *and* Edge-Cut *for N at level 0 can be solved in space $O(|N|^2)$.*

Proof. We first show how to check, for a given set $p \subseteq Q_N$, whether $[p]$ is a state in $[N]$, i.e., whether it is useful. As N only has useful states, there exists a path from $[p]$ to some accepting state in $[N]$. Thus it is enough to check whether there is a path from $[\{q_N^0\}]$ to $[p]$. This clearly can be done by a nondeterministic algorithm working in space $O(|N|)$. This algorithm would guess a word symbol by symbol, simulate $\mathcal{P}_N$ on the fly, starting from $\{q_N^0\}$ and test at each step whether the reached state q is equivalent to p, i.e., if $[p] = [q]$ (proof of Lemma 12). Thus, by Theorem 11, it can also be done by a deterministic algorithm working in space $O(|N|^2)$.

We now turn our attention to Edge. Let $(N, p, q, a, 0)$ be the input for Edge. We must decide if $[p] \xrightarrow{a} [q]$ is a transition in $[N]$. Since $[N]$ does not have useless states, this means that we should test two things:

- both $[p]$ and $[q]$ are states in $[N]$;
- $[\delta_{\mathcal{P}_N}(p, a)] = [q]$, where $\mathcal{P}_N$ is the power set automaton of N.

The former can be solved in $O(|N|^2)$, as we mentioned before. It remains to prove the latter. Given p and a, we can easily compute $\delta_{\mathcal{P}_N}(p, a)$ in space $O(|N|)$. According to Lemma 12, we can then decide if $[\delta_{\mathcal{P}_N}(p, a)] = [q]$ in space $O(|N|^2)$.

We omit the proof how to solve Acceptance and Edge-Cut. □

When we can solve Edge or Edge-Cut at a certain level, we can use it to make more complex tests.

Lemma 14. *Assume that we can solve* Edge *for N at level k in space $f(k, |N|)$. Then we can solve*

- Reachability *for N at level k in space $f(k, |N|) + O(|N|^2)$.*
- SameOrbit *for N at level k in space $f(k, |N|) + O(|N|^2)$.*
- InterOrbit *for N at level k in space $f(k, |N|) + O(|N|^2)$.*

Analogously, if we can solve Edge-Cut *for N at level k in space $f(k, |N|)$, then we can solve* Reachability-Cut, SameOrbit-Cut, *and* InterOrbit-Cut *for N at level k in space $f(k, |N|) + O(|N|^2)$.*

The next lemma allows us to do a single induction step that allows us to compute the structure of an automaton at level $k + 1$ if we can compute the automaton at level k.

Lemma 15. *Assume that we can solve* Edge *and* Acceptance *for N at level k in space $f(k, |N|)$. Furthermore, assume that $lev_k([N], [p])$ has no violation. Then we can solve* Edge *and* Acceptance *for N at level $k + 1$ in space $f(k, |N|) + O(|N|^2)$.*

The following lemma shows that we can compute properties of the level k automaton if we can decide some properties for all smaller levels. For technical reasons, we need the assumption that all lower level automata $\mathrm{lev}_i([N], [p])$ have no violation. This is because, otherwise, the level k automaton would be empty. The proof is basically a careful induction that puts together the previous lemmas.

Lemma 16. *Assume that for $0 \leq i \leq k - 1$ all automata $lev_i([N], [p])$ have no violation. Then all problems* Edge, Reachability, SameOrbit, InterOrbit, Acceptance *and* Edge-Cut, Reachability-Cut, SameOrbit-Cut, InterOrbit-Cut, *and* IsGate-Cut *for N at level k are in space $O((k + 1)|N|^2)$.*

Lemma 16 states that we can decide on-the-fly which transitions are present and which states are accepting in level k automata (if no violations occur in more shallow levels). Since these properties give the entire structure of the level k automata, we can now also test for violations on level k.

Lemma 17. *Assume that all automata $lev_i([N], [p])$ for $0 \leq i \leq k - 1$ have no violation. We can solve* Out-Cons-Violation, Acc-Cons-Violation *and* Orbit-Cons-Violation *for N at level k in space $O((k + 1)|N|^2)$.*

Proof. Let the input for Out-Cons-Violation be N and k. Then, an out-consistency violation occurs at level k of N if and only if there exist $p, q \subseteq Q_N$ such that all of the following hold:

- all automata $\mathrm{lev}_i([N], [p])$ for $0 \leq i \leq k - 1$ have no violation;
- both $[p]$ and $[q]$ are gates in $\mathrm{lev}_k([N], [p])_{S_k}$;
- both $[p]$ and $[q]$ are in the same orbit of $\mathrm{lev}_k([N], [p])_{S_k}$;

– there exist a symbol $a \in \Sigma$ and $[q']$ outside the orbit of $[p]$ in $\mathrm{lev}_k([N],[p])_{S_k}$ such that the transition $[p] \xrightarrow{a} [q']$ exists in $\mathrm{lev}_k([N],[p])_{S_k}$, but $[q] \not\xrightarrow{a} [q']$ does not.

By the lemma statement we know that all automata $\mathrm{lev}_i([N],[p])$ for $0 \leq i \leq k-1$ have no violation. According to Lemma 16 we can solve IsGate-Cut and SameOrbit-Cut for N at level k in space $O((k+1)|N|^2)$. We solve the last point by enumerating all $a \in \Sigma$ and states $q' \subseteq Q_N$ and testing whether

– a transition $[p] \xrightarrow{a} [q']$ exists and $[q] \xrightarrow{a} [q']$ does not exist;
– $[p]$ and $[q']$ are in different orbits.

Again by Lemma 16 Edge-Cut and SameOrbit-Cut for N at level k may be done in space $O((k+1)|N|^2)$. This concludes the proof for Out-Cons-Violation.

The proofs for Acc-Cons-Violation and Orbit-Cons-Violation are similar. □

We now have all the elements to prove our main result.

Theorem 18. DRE-Definability(*NFA*) *and* DRE-Definability(*RE*) *are PSPACE-complete.*

Proof. DRE-Definability(RE) is known to be PSPACE-hard [3]. Since an RE can be translated in polynomial time into an equivalent NFA, the lower bound also holds for NFAs.

Furthermore, the upper bound for REs follows from the upper bound for NFAs. We therefore show that DRE-Definability for an NFA N is in space $O(|N|^4)$, which proves the theorem. We assume w.l.o.g. that $|\Sigma| \leq |N|$. According to Theorem 10, a language $L(N)$ is not DRE-definable if and only if one of Out-Cons-Violation, Acc-Cons-Violation and Orbit-Cons-Violation occurs at some level k for N. Due to Theorem 9 we need to check this only for levels up to $|N|^2+2$, since the recursion depth of Algorithm 1 is never bigger than this.

We test whether there exists a violation at some level k for N, starting from level 0 and moving into higher and higher levels, up to level $|N|^2+2$. For every single level k the test can be done in space $O((k+1)|N|^2)$, according to Lemma 17. Note that during the application of the above lemma we know that the smaller levels do not contain any violation, since it was checked before, thus the assumptions in the lemma statements are fulfilled. Therefore the space needed for solving DRE-Definability(NFA) for N is bounded by $O((k+1)|N|^2)$ for $k = |N|^2+2$, i.e., bounded by $O(|N|^4)$, which finalises the proof for NFAs. □

4.2 Definability for RE(#)s

Next we show that testing DRE-Definability is EXPSPACE-complete when the input is given as a regular expression with counters. The upper bound is

immediate from Theorem 18 and the fact that we can translate an RE(#) into an RE of exponential size by unfolding the counters.

For the lower bound, we reduce from the exponential corridor tiling problem, which is defined as follows. An *exponential tiling instance* is a tuple $\mathcal{I} = (T, H, V, x_\perp, x_\top, n)$ where T is a finite set of tiles, $H, V \subseteq T \times T$ are the horizontal and vertical constraints, $x_\perp, x_\top \in T$, and n is a natural number in unary notation. A *correct exponential corridor tiling for* $\mathcal{I}$ is a mapping $\lambda : \{1, \ldots, m\} \times \{1, \ldots, 2^n\} \to T$ for some $m \in \mathbb{N}$, such that every of the following constraints is satisfied:

- the first tile of the first row is $x_\perp$: $\lambda(1,1) = x_\perp$;
- the first tile of the m-th row is $x_\top$: $\lambda(m,1) = x_\top$;
- all vertical constraints are satisfied: $\forall i < m, \forall j \leq 2^n, (\lambda(i,j), \lambda(i{+}1,j)) \in V$; and,
- all horizontal constraints are satisfied: $\forall i \leq m, \forall j < 2^n, (\lambda(i,j), \lambda(i,j{+}1)) \in H$.

Then, the EXP-Tiling problem asks, given an exponential tiling instance, whether there exists a correct exponential corridor tiling. The latter problem is known to be EXPSPACE-complete [27].

The full proof of the lower bound combines elements from the proof that DRE-definability for REs is PSPACE-complete [3] and that language universality for RE(#)s is EXPSPACE-complete [12].

Theorem 19. *Given a regular expression with counters r, the problem of deciding whether $L(r)$ is DRE-definable is EXPSPACE-complete.*

4.3 Definability for DFAs

As explained before, the DRE-definability problem is in polynomial time if the input is a minimal DFA [7]. As a final, minor result, we give an NLOGSPACE lower bound in this case.

Theorem 20. DRE-Definability($minDFA$) *is NLOGSPACE-hard.*

Proof (sketch). The reduction for the lower bound is from the reachability problem in directed acyclic graphs. This problem asks, given a DAG $G = (V, E)$, a source node s, and a target node t, whether t is reachable from s by a directed path. The DAG reachability problem is well-known to be NLOGSPACE-complete [17].

In this proof, we will use the fact that finite languages are always DRE-definable. (This can be checked through the Bkw-Algorithm which discovers immediately that all orbits are trivial.) For the reduction, let $G = (V, E)$, and nodes $s, t \in V$ be an instance of DAG reachability. We construct a minimal DFA D such that $L(D)$ is DRE-definable if and only if vertex t is reachable from vertex s in graph G.

We build $D = (Q, \Sigma, \delta, q_0, \{q_f\})$ from G as follows. The set Q of states is the disjoint union of the vertices V of G, plus two distinguished states q_0 (which

is D's initial state) and q_f (which is D's only accepting state). The alphabet Σ is defined as $(V \uplus \{q_0, q_f\})^2$. The transitions of D are defined as follows. Let $V = \{v_1, \ldots, v_n\}$ be the vertices of G.

- For each directed edge $(v_i, v_j) \in E$, the automaton D has the transition $\delta(v_i, (v_i, v_j)) = v_j$;
- for each vertex v_i, the automaton D has the transitions $\delta(q_0, (q_0, v_i)) = v_i$ and $\delta(v_i, (v_i, q_f)) = q_f$; and
- $\delta(t, (t, s)) = s$ is a transition of the automaton D.

As such, every transition has its unique label. This concludes the reduction. The reduction can be conducted in logarithmic space. □

5 Conclusions

We have pinned down the exact complexity of testing whether a regular expression can be determinized and considered additional variants of this problem. Our proof provides additional insights on such DRE-definable languages and on the decision algorithm of Brüggemann-Klein and Wood. An important open question is about the possible blow-up in the determinization process: Given a regular expression, what is the worst-case (unavoidable) blow-up when converting it to an equivalent deterministic one? At the moment, we know that an exponential blow-up cannot be avoided [21] but the best known upper bound is double exponential. While our proofs seem to give some insight in how to improve this upper bound, testing if a language is DRE-definable and actually constructing a minimal equivalent DRE are quite different matters. It is not yet clear to us how our techniques can be leveraged to obtain better upper bounds for this question.

Acknowledgement. We thank Wouter Gelade for bringing this problem to our attention.

References

1. M. Arenas, P. Barceló, L. Libkin, and F. Murlak. *Foundations of Data Exchange*. Book. To appear, 2013.
2. A. Balmin, Y. Papakonstantinou, and V. Vianu. Incremental validation of XML documents. *ACM Trans. on Datab. Syst.*, 29(4):710–751, 2004.
3. G. J. Bex, W. Gelade, W. Martens, and F. Neven. Simplifying XML Schema: effortless handling of nondeterministic regular expressions. In *ACM SIGMOD International Conference on Management of Data (SIGMOD)*, p. 731–744, 2009.
4. G. J. Bex, W. Gelade, F. Neven, and S. Vansummeren. Learning deterministic regular expressions for the inference of schemas from XML data. In *World Wide Web Conference (WWW)*, p. 825–834, 2008.
5. T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau. Extensible Markup Language XML 1.0 (fifth edition). W3C Recommendation, Nov. 2008.
6. A. Brüggemann-Klein and D. Wood. Deterministic regular languages. In *International Symposium on Theoretical Aspects of Computer Science (STACS)*, p. 173–184, 1992.

7. A. Brüggemann-Klein and D. Wood. One-unambiguous regular languages. *Inf. and Comput.*, 142(2):182–206, 1998.
8. H. Chen and L. Chen. Inclusion test algorithms for one-unambiguous regular expressions. In *International Colloquium on Theoretical Aspects of Computing (ICTAC)*, p. 96–110, 2008.
9. D. Colazzo, G. Ghelli, and C. Sartiani. Efficient inclusion for a class of XML types with interleaving and counting. *Inform. Syst.*, 34(7):643–656, 2009.
10. D. Fallside and P. Walmsley. XML Schema Part 0: Primer (second edition). World Wide Web Consortium, Oct. 2004.
11. W. Gelade, M. Gyssens, and W. Martens. Regular expressions with counting: Weak versus strong determinism. *SIAM J. Comput.*, 41(1):160–190, 2012.
12. W. Gelade, W. Martens, and F. Neven. Optimizing schema languages for XML: Numerical constraints and interleaving. *SIAM J. Comput.*, 38(5):2021–2043, 2009.
13. W. Gelade and F. Neven. Succinctness of the complement and intersection of regular expressions. In *ACM Trans. on Comput. Logic*, pages 4:1–19, 2012.
14. B. Groz, S. Maneth, and S. Staworko. Deterministic regular expressions in linear time. In *ACM Symposium on Principles of Database Systems (PODS)*, pages 49–60, 2012.
15. J.E. Hopcroft, R. Motwani, and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, 2007.
16. D. Hovland. Regular expressions with numerical constraints and automata with counters. In *International Colloquium on Theoretical Aspects of Computing (ICTAC)*, pages 231–245, 2009.
17. N. D. Jones. Space-bounded reducibility among combinatorial problems. *J. Comput. Syst. Sci.*, 11(1):68–85, 1975.
18. P. Kilpeläinen. Checking determinism of XML Schema content models in optimal time. *Inform. Syst.*, 36(3):596–617, 2011.
19. P. Kilpeläinen and R. Tuhkanen. One-unambiguity of regular expressions with numeric occurrence indicators. *Inform. and Comput.*, 205(6):890–916, 2007.
20. C. Konrad and F. Magniez. Validating XML documents in the streaming model with external memory. In *International Conference on Database Theory (ICDT)*, p. 34–45, 2012.
21. K. Losemann, W. Martens, and M. Niewerth. Descriptional complexity of deterministic regular expressions. In *Mathematical Foundations of Computer Science (MFCS)*, p. 643–654, 2012.
22. S. Maneth, A. Berlea, T. Perst, and H. Seidl. XML type checking with macro tree transducers. In *ACM Symposium on Principles of Database Systems (PODS)*, p. 283–294, 2005.
23. W. Martens and F. Neven. On the complexity of typechecking top-down XML transformations. *Theor. Comp. Sc.*, 336(1):153–180, 2005.
24. T. Milo, D. Suciu, and V. Vianu. Typechecking for XML transformers. *J. Comput. Syst. Sci.*, 66(1):66–97, 2003.
25. F. Neven and T. Schwentick. On the complexity of XPath containment in the presence of disjunction, DTDs, and variables. *Log. Meth. in Comp. Sc.*, 2(3), 2006.
26. L. Segoufin and V. Vianu. Validating streaming XML documents. *ACM Symposium on Principles of Database Systems (PODS)*, p. 53–64, 2002.
27. P. van Emde Boas. The convenience of tilings. In *Complexity, Logic and Recursion Theory*, pages 331–363. Marcel Dekker Inc., 1997.
28. P. T. Wood. Containment for XPath fragments under DTD constraints. *International Conference on Database Theory (ICDT)*, p. 297–311, 2003.